

Shared and Distributed Memory Parallelization of a Lagrangian Atmospheric Dispersion Model

David J. Larson, and J. S. Nasstrom

This article was submitted to Atmospheric Environment

May 11, 2001

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

Shared and Distributed Memory Parallelization of a Lagrangian Atmospheric Dispersion Model

David J. Larson and John S. Nasstrom
Atmospheric Science Division
Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, California 94551-0808, USA

Abstract

This paper describes parallelization of a 3-D Lagrangian stochastic atmospheric dispersion model using both distributed- and shared-memory methods. Shared-memory parallelism is implemented through the use of OpenMP compiler directives. Distributed-memory parallelism relies on the MPI message-passing library. One or both of the parallel modes can be used depending upon the requirements of the problem and the computational platform available. The distributed-memory version achieves a nearly linear decrease in execution time as the number of processors is increased. As the number of particles per processor is lowered, performance is limited by the decrease in work per processor and by the need to produce one set of output files. The shared-memory version achieves a speed-up factor of approximately 1.4 running on machines with four processors.

Key word index: Parallel processing, atmospheric dispersion, random walk dispersion model, OpenMP, MPI.

Introduction

The Lagrangian Operational Dispersion Integrator (LODI) is an atmospheric dispersion model developed for emergency response within the U.S. Department of Energy's National Atmospheric Release Advisory Center, NARAC (Nasstrom *et al.*, 2000). It solves the three dimensional advection-diffusion equation using a Lagrangian stochastic, Monte-Carlo method. A large number of independent particle trajectories are calculated and a contaminant air concentration is estimated from the spatial distribution of the particles at a particular time. LODI simulates a variety of physical processes including mean wind advection, turbulent diffusion, radioactive decay, first-order chemical reactions, wet deposition, gravitational settling, dry deposition, and buoyant or momentum dominated plume rise within the Lagrangian stochastic approach. LODI is a Fortran90 code and makes extensive use of the modern features available in this language, including array syntax, derived types, modules, pointers, and generic procedures.

Two considerations drove the parallel implementation of LODI. The first, not surprisingly, is execution speed. LODI runs in an operational environment so producing detailed high-resolution results as quickly as possible is critical to the emergency

response community served by NARAC. A second factor concerns prognostic plume dispersion calculations for scenario analysis. The regional weather forecasts that provide meteorological data to LODI can be improved using the techniques of ensemble forecasting. Ensemble forecasts explore the range of possible outcomes and provide a measure of the uncertainty associated with a single deterministic forecast (Toth and Kalnay, 1993; Sivillo, Ahlquist and Toth, 1997). Multiple LODI runs will be required to determine the different plume dispersion patterns generated from each member of the forecast ensemble. Thus execution speed is again a critical factor.

Luhar and Modi (1992) describe the implementation of parallel processing in a Lagrangian stochastic model that simulates vertical dispersion in the convective boundary layer. They note the inherently parallel nature of Lagrangian random-walk dispersion models and achieve a 14-fold improvement in execution time using 4096 processors in a SIMD (single instruction multiple data) machine of unique architecture using a non-standard, proprietary version of Fortran. In contrast, the work described below uses Fortran90, the standard message-passing interface MPI (Gropp, Lusk, and Skjellum; 1994), and the emerging standard for shared-memory processors, OpenMP (OpenMP Architecture Review Board, 2000), to produce a portable code with excellent scaling properties that runs on modern MIMD (multiple instruction multiple data) computers.

Numerical Model

LODI solves the following three-dimensional advection-diffusion equation:

$$\begin{aligned}
 \frac{\partial C}{\partial t} = & -\bar{u} \frac{\partial C}{\partial x} - \bar{v} \frac{\partial C}{\partial y} - \bar{w} \frac{\partial C}{\partial z} \\
 & + \frac{\partial}{\partial x} \left(K_x \frac{\partial C}{\partial x} \right) + \frac{\partial}{\partial y} \left(K_y \frac{\partial C}{\partial y} \right) + \frac{\partial}{\partial z} \left(K_z \frac{\partial C}{\partial z} \right) \\
 & + w_s \frac{\partial C}{\partial z} \\
 & - \Lambda C \\
 & - \lambda C \\
 & + Q
 \end{aligned} \tag{1}$$

where C is the mean air concentration of a species (e.g., units of kg m^{-3}), \bar{u} , \bar{v} and \bar{w} are the mean wind components (e.g., m s^{-1}) and K_x , K_y , and K_z are the eddy diffusivities (e.g., $\text{m}^2 \text{s}^{-1}$) in the x , y , and z directions, respectively, w_s is the absolute value of the gravitational settling velocity (e.g. m s^{-1}), Λ is the precipitation scavenging coefficient (e.g., s^{-1}), λ is the decay constant for radioactive decay or the rate constant for a first-order chemical reaction (additional terms are used for growth of due to decay from parent nuclides), and Q is the source term (e.g., $\text{kg s}^{-1} \text{m}^{-3}$). This is a conservation of species equation utilizing gradient diffusion theory (K theory) to model the turbulent flux of the

species. The mean wind components, \bar{u} , \bar{v} and \bar{w} , and the eddy diffusivities, K_x , K_y , and K_z , are meteorological fields residing on three-dimensional grids. The source term Q in Eq. (1) is specified using the parameters for the source geometry (initial spatial distribution of source material at any given time) and the total source material emission rate (mass or activity released per unit time).

The process described by the diffusion equation (1) can also be described by a stochastic differential equation in a Lagrangian reference frame (Durbin, 1983; Boughton, Delaurentis, and Dunn, 1987). For example, the equation for the displacement of a fluid particle in the vertical direction, may be written as follows:

$$\frac{dz}{dt} = \bar{w} + \sqrt{2K_z} \frac{dW}{dt} \quad (2)$$

where dW is a random variate with zero mean and variance dt , i.e.,

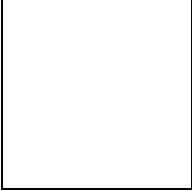
$$\langle dW \rangle = 0, \quad \langle dW^2 \rangle = dt$$

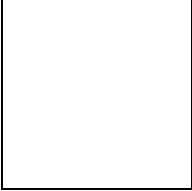
and is uncorrelated in time, i.e.,

$$\langle dW(t) dW(t') \rangle = 0 \quad \text{for } t \neq t'$$

$$\overline{\langle \dots \rangle}$$

An overbar, $\overline{\langle \dots \rangle}$, represents the ensemble average of a quantity. The form of the independent stochastic differential equations for the displacements in the x and y directions are identical with the exceptions that the terms with the spatial derivative of the eddy diffusivities K_x and K_y are assumed to be negligible and there is no contribution of the gravitational settling speed to the displacement. These equations describe the ensemble of possible particle trajectories. Each trajectory represents one realization. Time integration of Eq. (2), and the corresponding horizontal equations, provides a means to

calculate the  trajectory of a particle, given its initial position. This provides a basis for Lagrangian, Monte Carlo numerical simulations in which a sample of

N independent particle trajectories, , originating from a source are used to estimate the probability density function for the particle position at time t , and, given the spatial and temporal distribution of species mass emitted from the source, to estimate the ensemble mean concentration $\bar{C}(x, y, z, t)$.

LODI solves Equation (2) using the methods described by Leone, *et al.* (1997) and Ermak and Nasstrom (2000). Particle displacements due to the mean wind are calculated using either second- or fourth-order Runge-Kutta methods (Leone *et al.*, 1997). The displacement of a particle due to turbulent diffusion is calculated using a skewed, non-Gaussian particle position probability density function (Ermak and Nasstrom, 2000). Diffusive displacements are determined by random sampling from this function. This non-Gaussian method is of higher order in time than a previously used Gaussian method. Ermak and Nasstrom (2000) compared numerical simulation results to analytic solutions of the diffusion equation, and showed that their method is significantly more efficient than using a Gaussian particle position distribution.

The general algorithm for the LODI code is:

- 1) Initialization: read in grid coordinates for meteorological data and concentration/deposition data, read in meteorological data, initialize source variables.
- 2) Advance particles in time: determine time-step for each particle, calculate the diffusive displacement, and calculate displacement due to advection by the mean wind.
- 3) Calculate particle contributions to gridded air concentration and ground deposition fields.
- 4) Generate output

Note that step (2) can also include calculating radioactive growth and decay, the gravitational settling velocity, buoyant and momentum-dominated plume rise, dry deposition, as well as precipitation scavenging.

Distributed Memory Parallelism

LODI is well suited to the distributed memory parallel model due to the independence of individual particle trajectories. On a distributed memory computer each processor, or central processing unit (CPU), has its own dedicated computer memory. We distribute the

simulation particles across the processors in an ordered way. The first particle is assigned to the first processor, particle two to the second processor, and so on until the N th particle is assigned to the N th processor. Further particles start the cycle over so particle number $N+1$ is assigned to the first processor, $N+2$ to the second and so on.

Each processor has a complete copy of the gridded meteorological data and the grids required to calculate the various contaminant concentrations requested by the user. A grid is used to define the sampling volumes over which the concentrations are calculated. The time-integrated (and time-average) concentration is calculated by using the contaminant mass associated with a particle and the time spent in the sampling volume, summing over all particles passing through the sampling volume. Particle contributions to time-integrated and time-averaged gridded concentrations are calculated separately on each processor. However, at time intervals specified by the user, concentration contributions from particles on all processors must be assembled, either for output or for determining the peak time-average concentration within user-specified averaging time intervals in between output times. Calculating the peak average concentration requires assembling all concentration information on one processor at the end of each averaging time. This is accomplished by passing the concentration information to the first processor using MPI routines to send and receive data. Global reductions, which combine values from all processors and distribute the result back to all processors, are also used in order to determine the number of active simulation particles and the total mass released on particles for simulation status reporting and error checking. In order to minimize the number of output files, concentration and particle data from all processors are shipped to the root processor for output at user-specified intervals.

In addition to distributing particles across the processors, individual processors must have independent random number streams in order to achieve independence of all particle trajectories in a simulation. There is a vast literature on parallel random number issues, for examples see Percus and Kalos (1989) and De Matteis and Pagnutti (1995). We implemented a combination generator by Wollan (1992) in LODI that achieves distinct random number streams through a congruential component and a long period (10^{25}) through a Fibonacci component. The method is portable, consisting of a self-contained Fortran90 module, and is useful for up to one thousand processors.

The problem size that the distributed memory version of LODI can run is limited by the memory available to each processor. The IBM ASCI (Accelerated Strategic Computing Initiative) Blue machine at Lawrence Livermore National Laboratory has four processors per node sharing up to two and one half gigabytes of memory. Since each processor has a complete copy of all the meteorological and concentration grid data, using all the processors on a node results in four copies of these data being stored on the node. This puts an upper bound on the problem size one can run while maximizing the system resources by using all the processors on a node.

Shared Memory Parallelism

As an alternative to the distributed memory implementation, a shared memory version of LODI was also developed. On a shared memory computer all processors use the same memory. We use the OpenMP application program interface to achieve shared memory parallelism (OpenMP Architecture Review Board, 2000). OpenMP supports a fork-join model of parallel execution through the use of compiler directives, library routines, and environment variables. In the fork-join model a single process, known as the master thread, executes the program sequentially until it encounters a parallel directive. The master thread then creates a team of threads that execute the specified code in parallel. Upon completion of the parallel region, the thread team synchronizes and only the master thread continues execution.

In order to make effective use of OpenMP compiler directives, the main time-step loop in LODI was written to perform each particular computation (e.g., calculating the diffusive displacement) for a block of particles at a time, before proceeding to the next computation (e.g., determining the mean wind advection velocity at the particle position). OpenMP supports fine-grained parallelism in which the work associated with each loop can be spread across multiple processors. In a typical LODI run approximately eighty five percent of the run time is spent in the subroutine that updates the particle position, so the majority of the shared memory parallelism occurs in this routine. This routine calculates the plume rise velocity, the gravitational and dry deposition velocities, the precipitation scavenging coefficient, the time step restrictions due to the grid cell size, the horizontal and vertical eddy diffusivities, the diffusive displacement and the displacement due to the mean wind advection. The number of particles comprising a particle block can be varied in order to obtain maximum performance on today's cache-based computer architectures. Table 1 gives performance results for a test problem run with fifty thousand simulation particles using four threads.

Based on these results, we typically set the number of particles per block to 501. This choice yields good results on our suite of test problems run on both the ASCI Blue machine and the NARAC Compaq Alpha systems.

Eleven out of a total of fourteen major loops in the particle position subroutine were amenable to parallelization by OpenMP compiler directives. Several of the remaining loops had potential memory conflicts due to the use of pointers that depend on the value of the loop index. OpenMP directives were also used in a few loops in the main LODI control routine.

Table 1: Execution time as a function of the number of particles per block.

Number of particles per block	Time (s)
32	510
128	312
512	267
2048	287

The use of shared memory parallelism should allow a larger problem to fit on a computational node since only one copy of the meteorological and concentration grid data is required per node. As the results in the following section demonstrate, the parallel efficiency of the OpenMP version is not as high as that of the MPI version. However, higher spatial resolution simulations may require the use of the shared memory version of LODI.

Results

Figure 1 shows the run time on the ASCI Blue machine at LLNL versus number of processors for a benchmark problem run with fifty thousand particles. The squares are the MPI LODI results and the line denotes linear scaling, determined by dividing the execution time for a single processor run by the corresponding number of processors.

As expected, LODI nearly achieves the optimum of linear speed up as long as the load per processor overcomes the latency resulting from inter-processor communication. As the number of processors increases, the results demonstrate the well known fall off in parallel efficiency as the computational load per processor is reduced. The efficiency increases at the higher end of the scale when more particles are used (not shown).

Results for running the benchmark problem using one million simulation particles are shown in Table 2. Here we define speedup as the ratio of the execution time for the baseline case to the execution time for a given test case. The OpenMP standard allows you to specify how the loop iterations are divided (scheduled). Among the threads `OMP guided` refers to the OpenMP version of LODI using the guided schedule, which divides the loop iterations among the threads so that the size of each successive piece is exponentially decreasing. There are a variety of choices for the scheduling directive — we chose `guided` because it produced the best results in our implementation. The MPI overhead referred to above is a small performance penalty incurred on the IBM ASCI Blue machine when running an MPI code on all four CPUs within a given node compared to using one processor per node and four nodes. The intra-node processor communication is slightly less efficient than inter-node communication due to the machine architecture. However, this penalty is not large enough to justify wasting CPU cycles by using only one processor per node. The OpenMP version running with four CPUs is limited to a speedup of about 1.4 due to the limited number of loops benefiting from parallel execution. If perfect parallel execution of the position update subroutine could be achieved, a speedup of about 2.75 would be expected, based on dividing the time spent in this subroutine by four and comparing the total run time with the baseline case.

Table 2: Execution time for the shared- and distributed-memory parallel versions of LODI for several test cases

Test case	Time (s)	Comments
-----------	----------	----------

Baseline case, 1 node, 1CPU	3920.	
MPI, 4 nodes, 4 CPUs	987.	MPI speedup = 3.97
MPI, 1 node, 4 CPUs	1007.	Pure MPI overhead = 2%
OMP guided, 1 node, 4 CPUs	2715.	OMP speedup = 1.44
MPI + OMP guided, 5 nodes, 20 CPUs	535.	Speedup = 7.3

LODI can also be run using both distributed and shared memory models. In this case MPI is used for internode communication and OpenMP is used for parallel intranode execution. This yields a speedup equal to the typical MPI speedup multiplied by the typical OpenMP speedup, or a factor of 7.3 in the test case using 5 nodes with a total of 20 CPUs.

Conclusions

We have developed a parallel version of the Lagrangian, Monte-Carlo atmospheric dispersion code LODI using both the shared and distributed memory models. The version (MPI, OpenMP, or both) is selected when compiling the code. The MPI version achieves nearly linear speedup as long as the computational load is sufficient to overwhelm the air concentration output processing (this is nearly always the case). The OpenMP version achieves a speedup of 1.4 for typical problems run on machines with four CPUs per node. Additional speed could possibly be achieved by further restructuring, for example by eliminating the dependence of pointer assignments on the loop index through the use of sorting, but the advantages of this haven't been sufficiently compelling to date.

Running the combined version (MPI and OpenMP) can be useful on machines with more than one CPU per node (e.g., the ASCI Blue machine) using MPI for inter-node communication and OpenMP for intra-node parallelism. The memory requirements usually dictate which version is most appropriate. A problem so large that four complete copies of the required data will not fit on a node, as required by the MPI version, may be successfully run using the OpenMP version.

Acknowledgments This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

- Boughton, B. A., J. M. Delaurentis, and W. E. Dunn (1987). A stochastic model of particle dispersion in the atmosphere, *Boundary Layer Meteorology*, **40**, 147-163.
- De Matteis, A. and S. Pagnutti (1995). Controlling correlations in parallel Monte Carlo, *Parallel Computing*, **21**, 73-84.
- Durbin, P.A. (1983). *Stochastic differential equations and turbulent dispersion*. NASA Reference Publication 1103, 69 pp. (Available from NTIS as N8322546.)

- Ermak, D. L. and J. S. Nasstrom (2000). A Lagrangian stochastic diffusion method for inhomogeneous turbulence, *Atmospheric Environment*, **34**, 1059-1068.
- Gropp, W., E. Lusk, and A. Skjellum (1994). *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, Massachusetts.
- Leone, J. M. Jr., J. S. Nasstrom, and D. Maddix (1997). A first look at the new ARAC dispersion model. Preprint, *American Nuclear Society Sixth Topical Meeting on Emergency Preparedness and Response*, San Francisco, CA, April 1997, American Nuclear Society, Inc. La Grange Park, IL.
- Luhar, A.K. and J. J. Modi (1992) Parallel processing of a random-walk model of atmospheric dispersion, *Atmospheric Environment*, **26A**, 3055-3059.
- Nasstrom, J.S., G. Sugiyama, J.M. Leone, Jr., and D.L. Ermak (2000). A real-time atmospheric dispersion modeling system, Preprint, *Eleventh Joint Conference on the Applications of Air Pollution Meteorology*, Long Beach, CA, Jan. 9-14, 2000. American Meteorological Society, Boston, MA, 84-89.
- OpenMP Architecture Review Board (2000). OpenMP Fortran Application Program Interface, Version 2.0 (www.openmp.org).
- Percus, O. E. and M. L. Kalos (1989). Random number generators for MIMD parallel processors. *Journal of Parallel and Distributed Computing*, **6**, 477-497.
- Sivillo, J. K., J. E. Ahlquist and Z. Toth (1997). An ensemble forecasting primer, *Weather and Forecasting*, **12**, 809-818.
- Toth, Z., and E. Kalnay (1993). Ensemble forecasting at NMC: The generation of perturbations. *Bulletin of the American Meteorological Society*, **74**, 2317-2330.
- Wollan, P.C. (1992). A portable random number generator for parallel computers. *Communications in Statistics- Simulation*, **21**, 1247-1254.

Figure captions

Fig. 1. Execution time versus number of processors for the distributed memory (MPI) version of LODI.